Week8: Improving Networks with Regularization (cont.)

SETH HUANG

- PRESENTATION AND PAPER DETAILS

WE HAVE DISCUSSED THE FOLLOWING:

- Python programming language
- Neural network structure
- Weights, nodes, neurons
- Using the Python codes for MNIST (written digits) model training
- How to judge training data and test data
- The math behind training the model
- Regularization (L2) and why it helps
- Other regularization techniques (expanding data, L1, drop-out techniques)

=> We are getting very close to deep-learning

ARTIFICIALLY EXPANDING DATA

First, warm up:

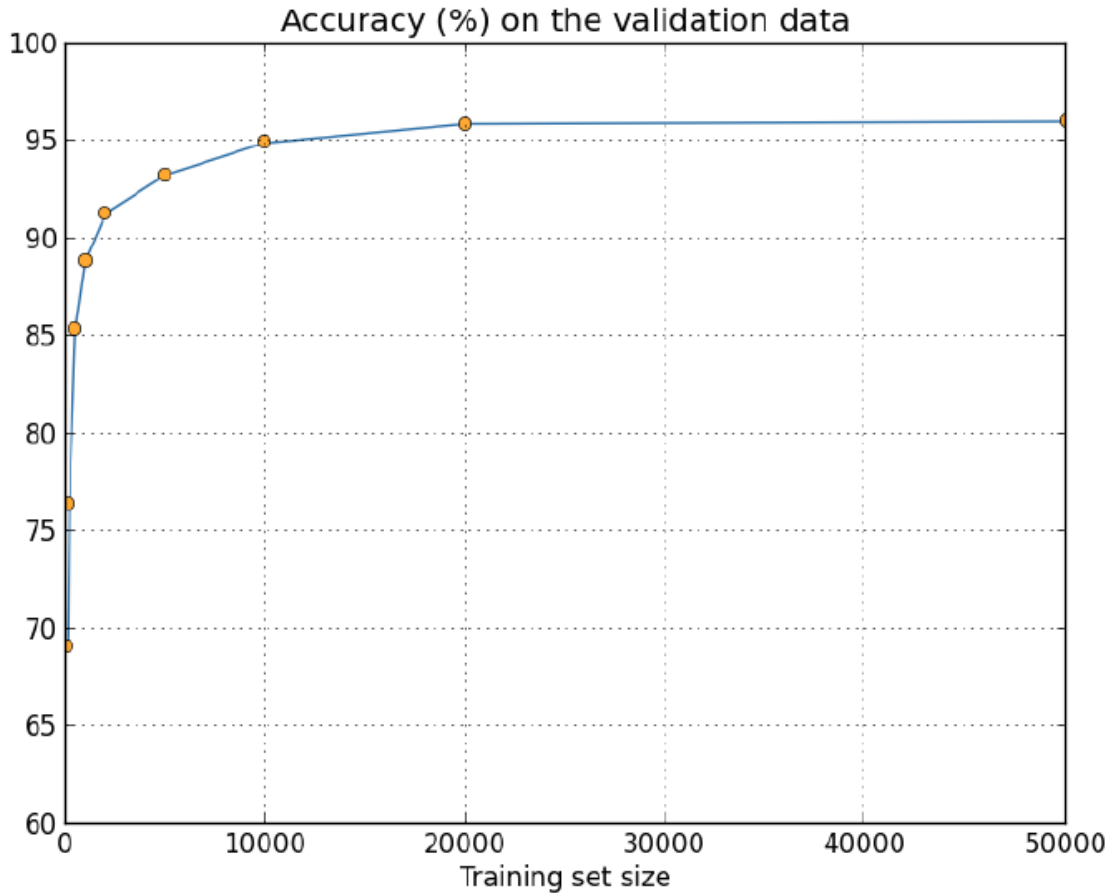
Eta = 0.5, 30 hidden neurons, minibatch = 10

Try out codes with 1000 images (10 epochs) (lmbda = 0.1)

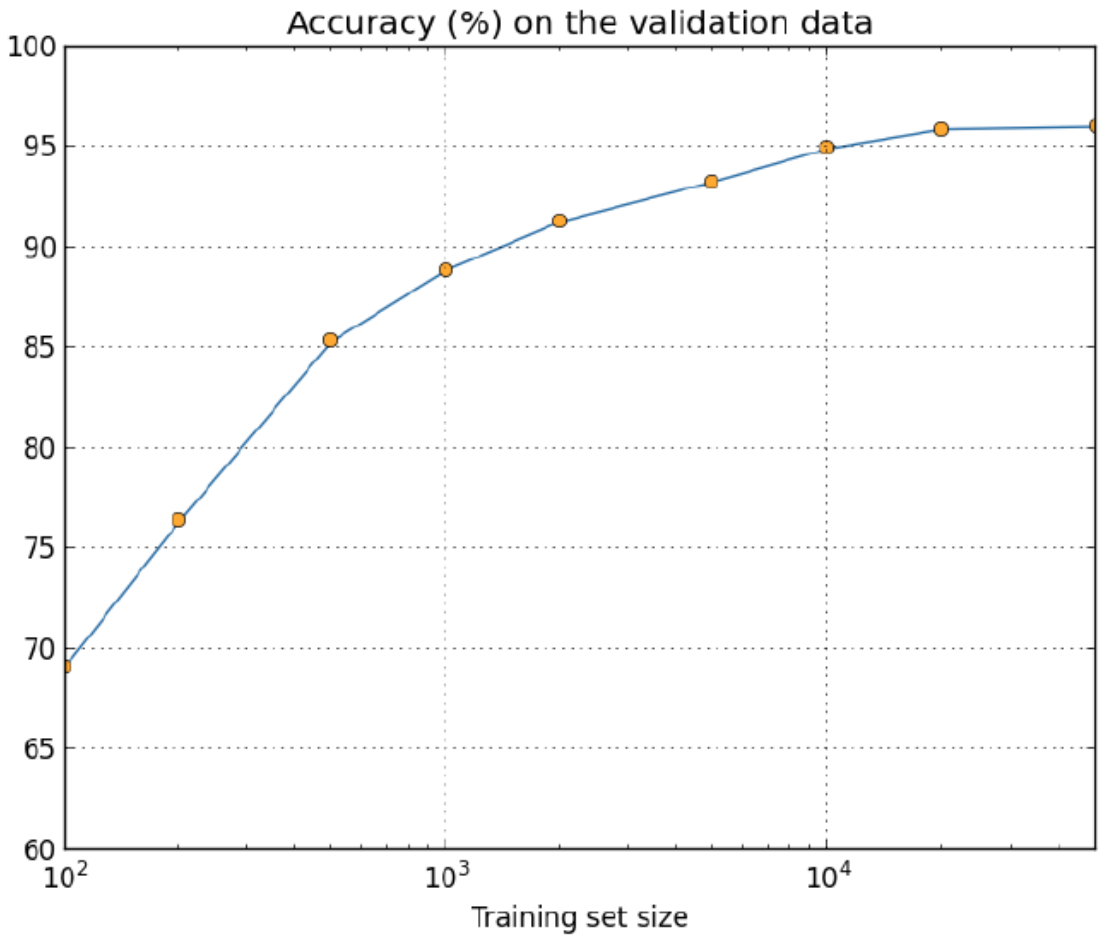
Try out codes with 5000 images (10 epochs) (adjust)

Try out codes with 50000 images (10 epochs) (adjust)

MOREDATA.PY


Accuracy (%) on the validation data
100
95
90
85
80
75
70
65
60
0
10000
20000
30000
40000
50000
Training set size

LOG (TRAINING SIZE)



THE TAKEAWAYS

What happened the learning accuracy as we increase the training size?

What happened to the rate of improvements?

What if we use millions of training data?

OBTAINING MORE DATA IS EXPENSIVE

But we can get more data by changing the original ones slightly

We can stretch, rotate, move around, and the more we do that, the better (more robust) the results would be.

THERE ARE PAPERS DISCUSSING VARIOUS TECHNIQUES. FOR EXAMPLE:

*Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis, by Patrice Simard, Dave Steinkraus, and John Platt (2003).

With simple network and cross-entropy, they obtained 98.4% Accuracy with 800 hidden neurons.

Then they skew the images to obtain more data => 98.9%

Then they use something called "elastic distortion," a randomized oscillation that simulate random muscle movements of one's hand => 99.3%

Error goes from 1.6% to 1.1% to 0.7%
That's ~30% improve and then another 35% of improvement!

THIS TECHNIQUE IS IMPORTANT AND CAN BE USED FOR MANY AREAS

- Images of other kinds
- Speech recognition

⇒ Speech who is slower or faster

⇒ Speech with background noises

Can you think of anything else?

THIS TECHNIQUE IS IMPORTANT AND CAN BE USED FOR MANY AREAS

- Images of other kinds
- Speech recognition

⇒ Speech who is slower or faster

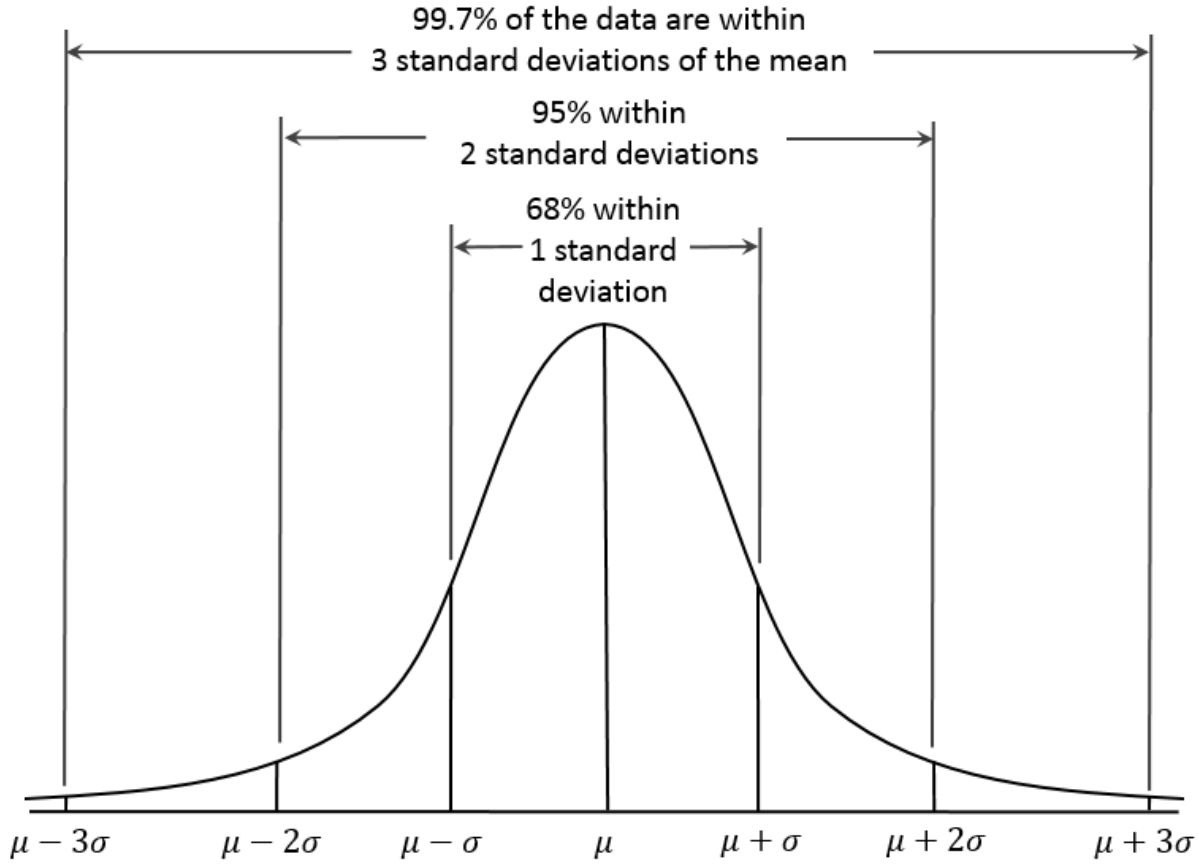
⇒ Speech with background noises

Can you think of anything else?

WEIGHT INITIALIZATION

BEFORE, IT'S ALWAYS BEEN RANDOMLY INITIALIZED

Normally distributed with mean 0 and std of 1. Can you see how this works?

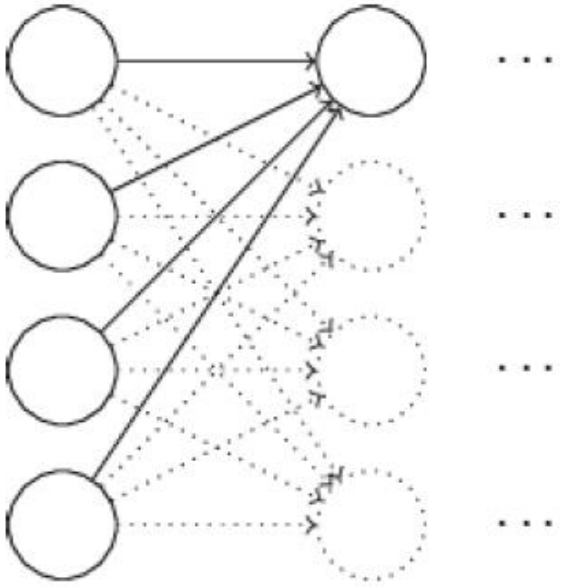


WE CAN TRY TO DO IT BETTER THAN NORMAL DISTRIBUTION

The following will be a bit complex, so please follow more carefully.

First, let's start with 1000 input neurons => Some data that has 1000 inputs.

Focus on the input neurons TO THE FIRST HIDDEN NEURON



AS YOU KNOW, IN A FEED-FORWARD NETWORK…

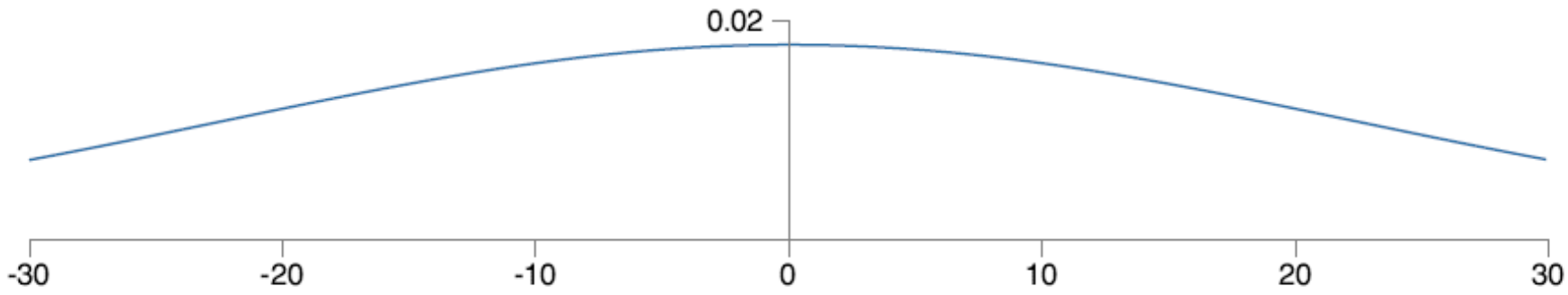
$$\text{weighted sum } z = \sum_j w_j x_j + b$$

Assume 50% of input says 1 (based on some data), and 50% of input says 0.

That means z is a sum of 500 zeros and 500 ones. We know that zeros are just zeros (ignoring the bias). So that means, since weights are normally-distributed and we have 500 ones, the distribution of z is:

Mean = 0
Standard deviation = $\sqrt{501} \approx 22.4$

THIS MEANS THE DISTRIBUTION OF Z IS:

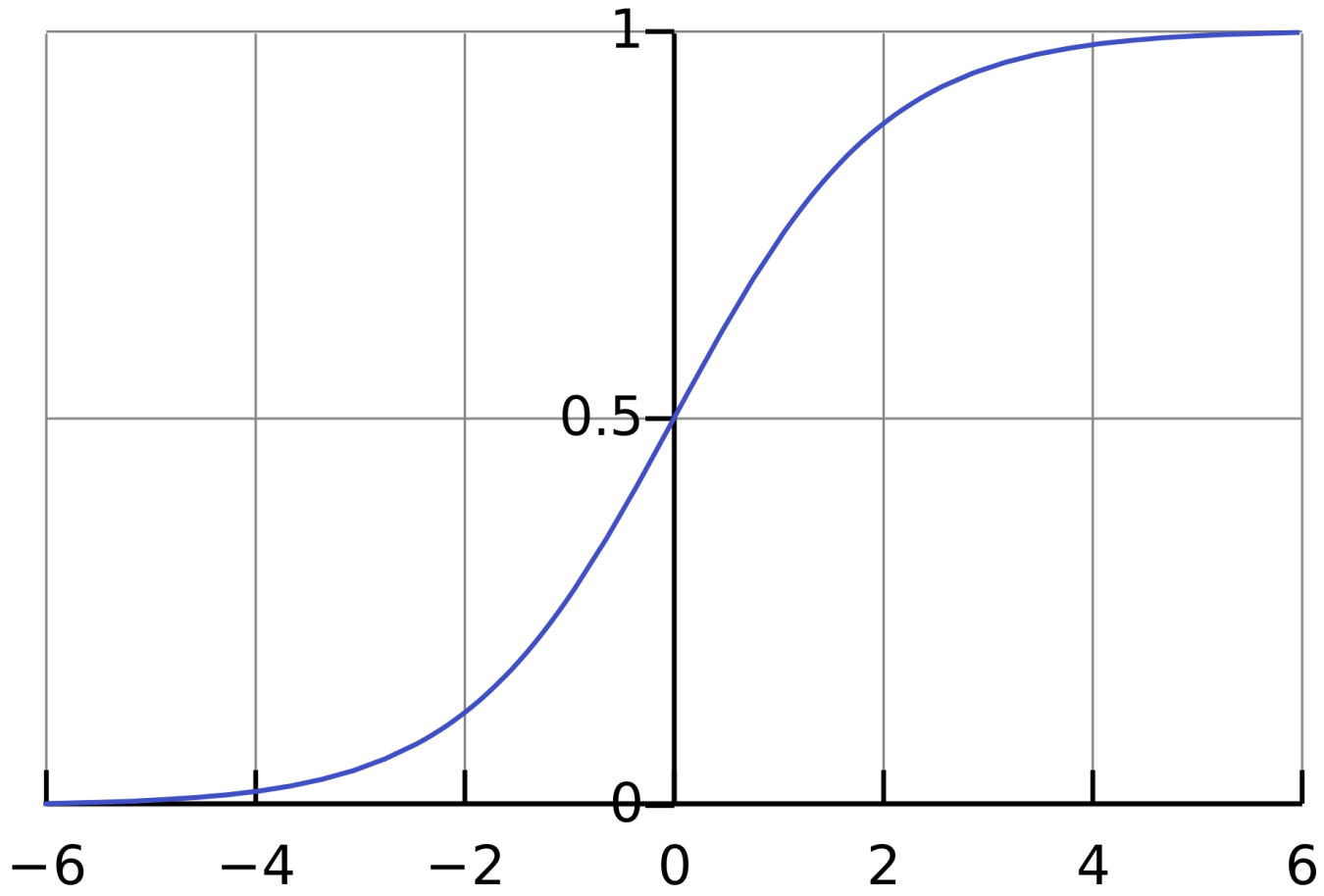


This is a super fat-tailed distribution with huge standard deviation

This means it is easy for z to be very large OR very small

What does that mean in terms of activation function like Sigmoid?

REMEMBER THE SIGMOID FUNCTION AND "WEIGHT SATURATION"



NOTE THAT CROSS-ENTROPY DOES NOT SOLVE THIS PROBLEM, SO WE NEED TO INITIALIZE WEIGHTS TO AVOID SATURATION

Again, what specific problem are we trying to solve?

So, if, as the number of input neurons gets larger, we get big standard deviation of Z. Can you think of any ideas?

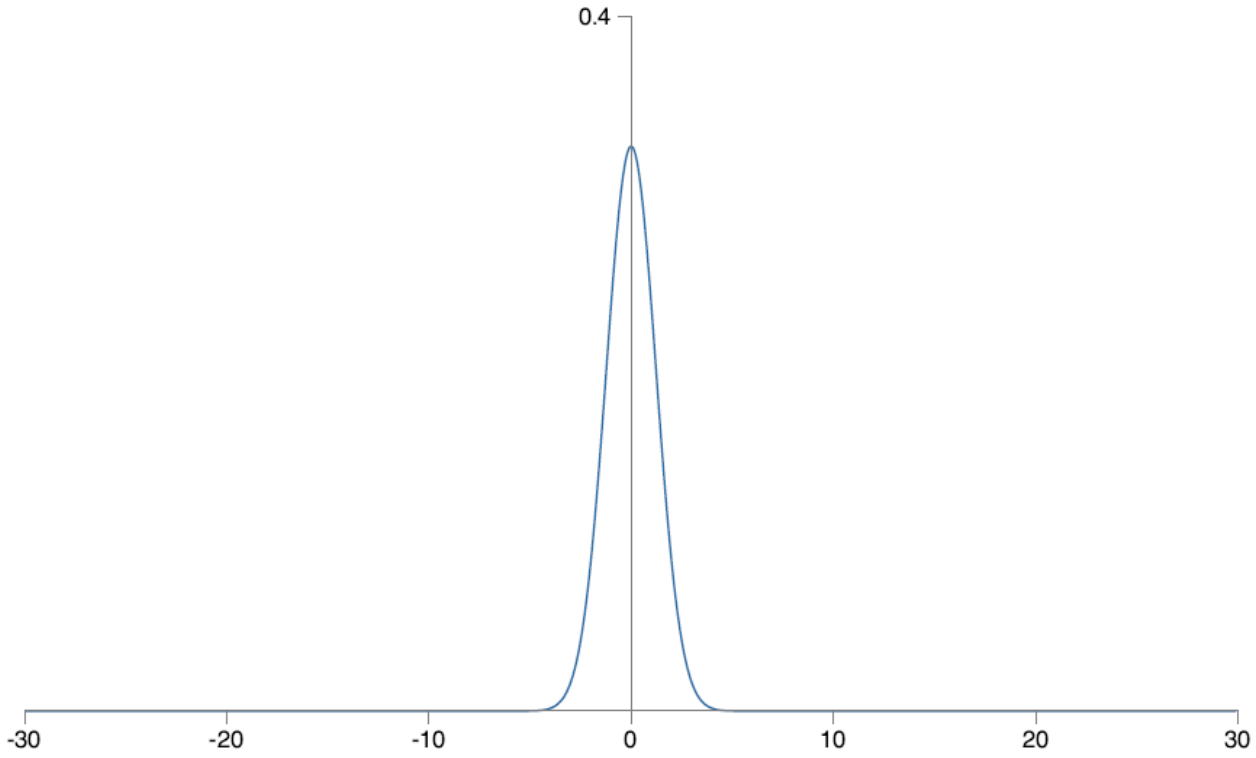
Remember, how did we initialize the weights before?

THE ANSWER IS SIMPLE: WE SCALE THE INITIALIZATION BASED ON THE NUMBER OF INPUTS

Initialize weights to be
Mean = 0
Standard deviation = $1/\sqrt{n_{\text{in}}}$.

The bigger the input size =>
The sharper the distribution is (and the starting points would be closer to zero)

FOR THE SAME EXAMPLE (1000 NEURONS, 500 OF ONES AND 500 OF ZEROS)



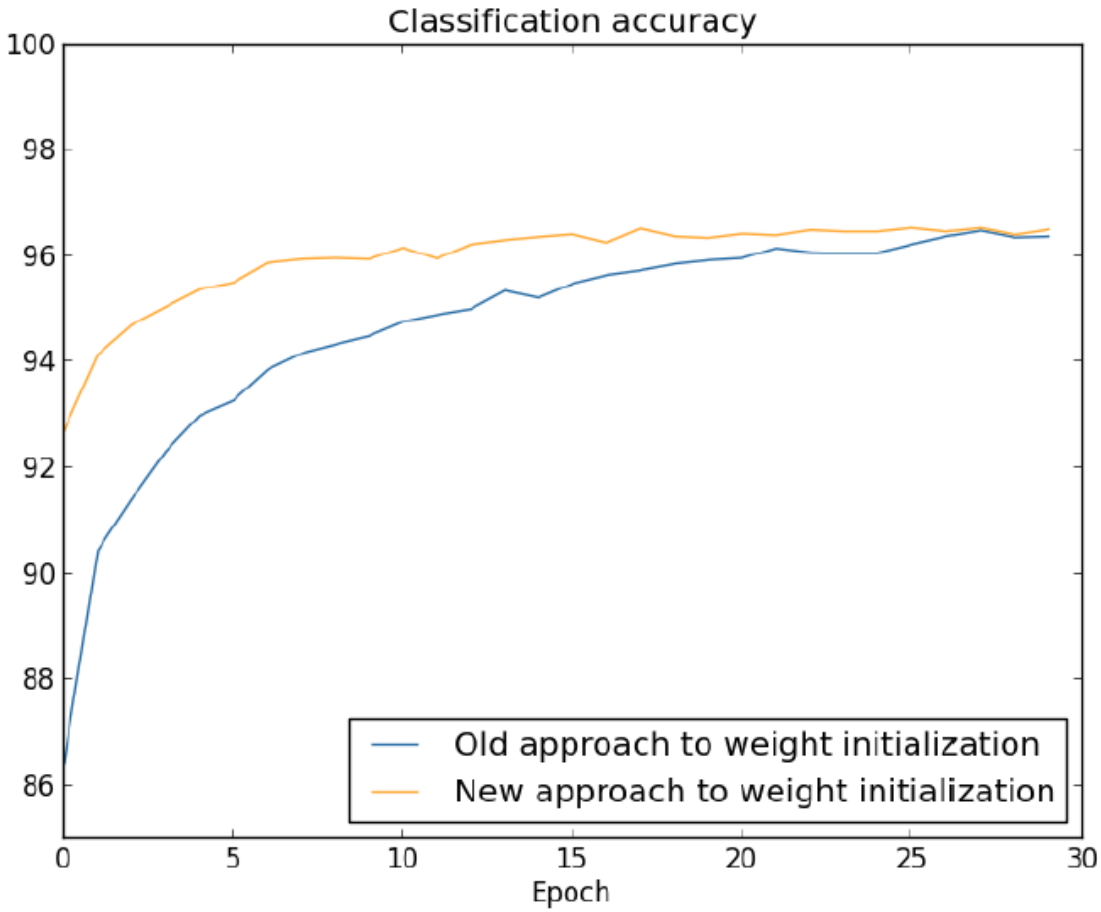
THE OLD APPROACH

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data, 30, 10, 0.1, lmbda = 5.0,
... evaluation_data=validation_data,
... monitor_evaluation_accuracy=True)
```

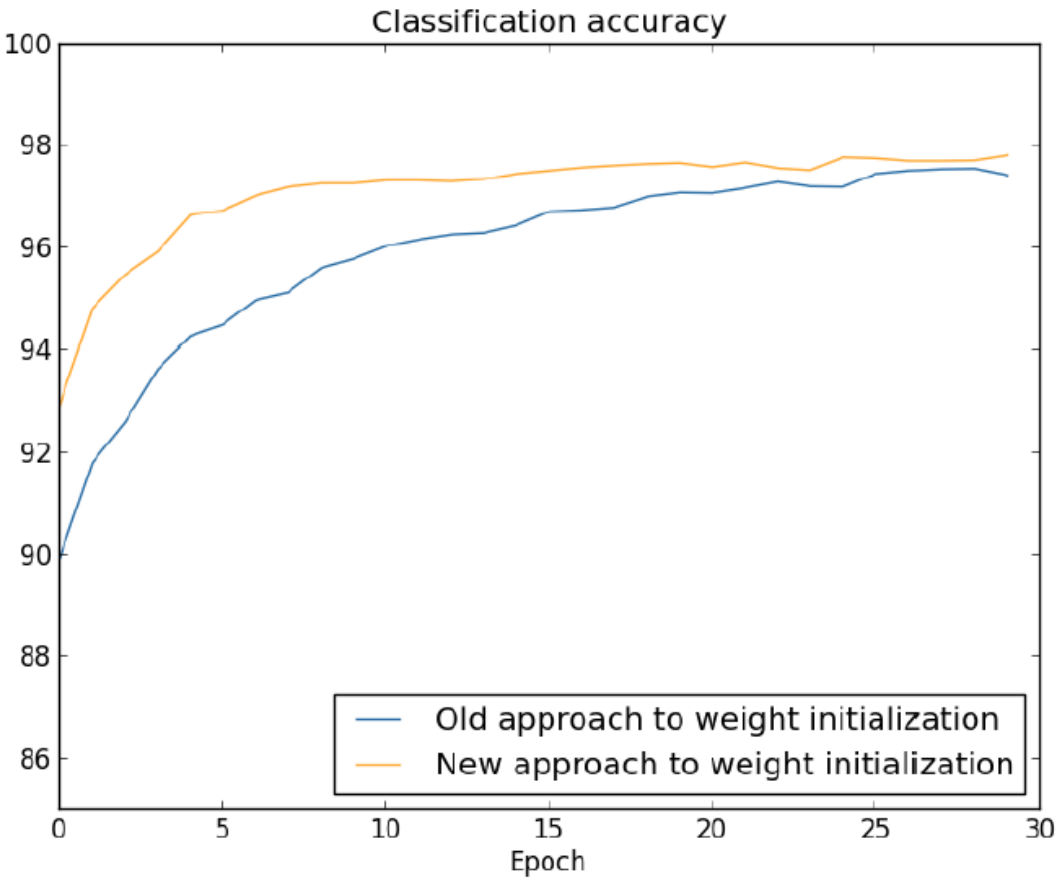
THE NEW APPROACH, CHECK THE FIRST FEW EPOCHS

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data, 30, 10, 0.1, lmbda = 5.0,
... evaluation_data=validation_data,
... monitor_evaluation_accuracy=True)
```

THE DIFFERENCE (30 NEURONS)



THE DIFFERENCE (100 NEURONS)



KNOWING ALL THAT, HOW DO YOU CHOOSE THE HYPER-PARAMETERS?

What are the hyper-parameters that you have to decide on even for the simplest network? What do they do?

Note that in the example, everything was carefully chosen for you already

WHAT IF YOU BEGIN WITH THIS?

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10])
>>> net.SGD(training_data, 30, 10, 10.0, lmbda = 1000.0,
... evaluation_data=validation_data, monitor_evaluation_accuracy=True)
Epoch 0 training complete
Accuracy on evaluation data: 1030 / 10000

Epoch 1 training complete
Accuracy on evaluation data: 990 / 10000

Epoch 2 training complete
Accuracy on evaluation data: 1009 / 10000

...

Epoch 27 training complete
Accuracy on evaluation data: 1009 / 10000
```

MORE CHOICES:

- Type of cost function?
- Number of layers (for deep learning)
- The "type" of layers (which you will learn)
- The type of activation function (Sigmoid, TanH, ReLU?)
- Number of minibatches
- And of course, weight initializer

And that's after you deal with:

- Data collection
- Data organization
- Making samples
- Dividing them properly
- Pre-processing (beyond the scope of this class)

HERE, I WILL DISCUSS SOME BASIC FRAMEWORK TO DEAL WITH THESE BASIC PROBLEMS (WHICH YOU SHOULD USE FOR YOUR PAPER)

You should begin by making sure the data labels are more or less evenly distributed:

=> 2000 of zeros, 2000 of ones, 2000 of twos …. 2000 of nines

For MNIST, you are trying to get the machine to identify 10 digits. But you can pick 2000 of zeros and 2000 of ones first.

=> See if you can get non-trivial results first (>50%)

=> Must faster to train

You can also pick a small subset of data to train first.

IF YOU WANT TO SEE WHAT "TRIVIAL" LOOKS LIKE WITH LMBDA = 1000:

```
>>> net = network2.Network([784, 10])
>>> net.SGD(training_data[:1000], 30, 10, 10.0, lmbda = 1000.0, \
... evaluation_data=validation_data[:100], \
... monitor_evaluation_accuracy=True)
Epoch 0 training complete
Accuracy on evaluation data: 10 / 100
```

```
Epoch 1 training complete
Accuracy on evaluation data: 10 / 100

Epoch 2 training complete
Accuracy on evaluation data: 10 / 100
...
```

IF YOU WANT TO SEE WHAT IT LOOKS LIKE WITH LMBDA = 20:

```
>>> net = network2.Network([784, 10])
>>> net.SGD(training_data[:1000], 30, 10, 10.0, lmbda = 20.0, \
... evaluation_data=validation_data[:100], \
... monitor_evaluation_accuracy=True)
Epoch 0 training complete
Accuracy on evaluation data: 12 / 100

Epoch 1 training complete
Accuracy on evaluation data: 14 / 100

Epoch 2 training complete
Accuracy on evaluation data: 25 / 100

Epoch 3 training complete
Accuracy on evaluation data: 18 / 100
```

IF YOU WANT TO SEE WHAT "TRIVIAL" LOOKS LIKE WITH ETA (LEARNING RATE) = 100:

```
>>> net = network2.Network([784, 10])
>>> net.SGD(training_data[:1000], 30, 10, 100.0, lmbda = 20.0, \
... evaluation_data=validation_data[:100], \
... monitor_evaluation_accuracy=True)
Epoch 0 training complete
Accuracy on evaluation data: 10 / 100

Epoch 1 training complete
Accuracy on evaluation data: 10 / 100

Epoch 2 training complete
Accuracy on evaluation data: 10 / 100

Epoch 3 training complete
Accuracy on evaluation data: 10 / 100
```

IF YOU WANT TO SEE WHAT "TRIVIAL" LOOKS LIKE WITH ETA (LEARNING RATE) = 1:

```
>>> net = network2.Network([784, 10])
>>> net.SGD(training_data[:1000], 30, 10, 1.0, lmbda = 20.0, \
... evaluation_data=validation_data[:100], \
... monitor_evaluation_accuracy=True)
Epoch 0 training complete
Accuracy on evaluation data: 62 / 100

Epoch 1 training complete
Accuracy on evaluation data: 42 / 100

Epoch 2 training complete
Accuracy on evaluation data: 43 / 100

Epoch 3 training complete
Accuracy on evaluation data: 61 / 100

...
```

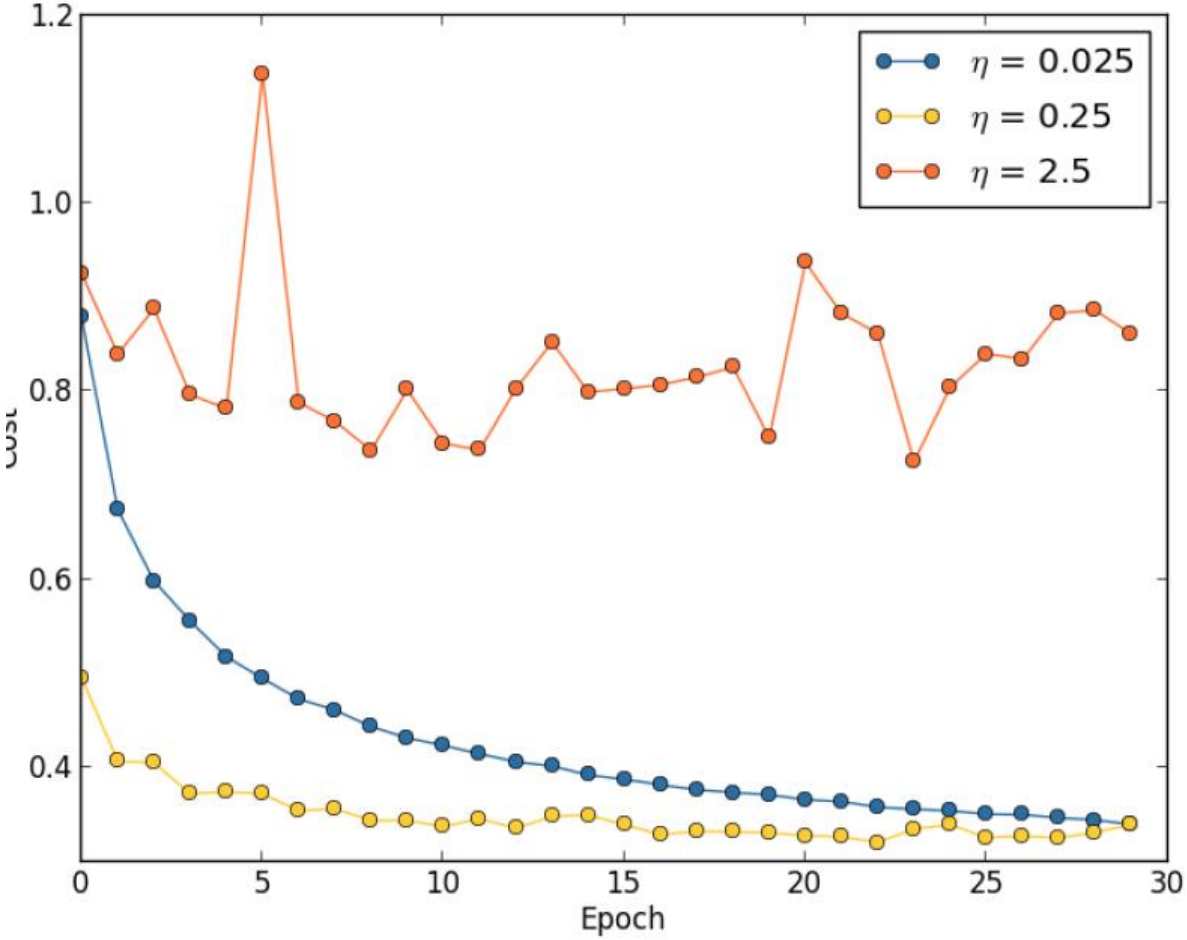
BUT EVEN THIS IS TOO POSITIVE

The key is to get fast feedbacks from your machine.
Always start simple, and build your model around the initial results.

For example, if we use our data to examine different learning rate, you can get a feeling what the learning rate should be.

For example, you can use eta = [0.025, 0.25, 2.5]

"TRAINING COST" PROGRESSION



"TRAINING COST" PROGRESSION